CMP_SC 8001 - Introduction to Secure Multiparty Computation

Defining Multi-Party Computation

Wei Jiang

Department of Electrical Engineering and Computer Science

University of Missouri

Outline

Outline

1. Notations and Conventions
   - Common Notations
   - Basic Definitions
   - Basic Primitives
2. Security of Multi-Party Computation
   - Real-Ideal Paradigm
   - Semi-Honest Security
   - Malicious Security
   - Hybrid Worlds and Composition
3. Specific Functionalities of Interest
   - Oblivious Transfer
   - Commitment
   - Zero-Knowledge Proof

Common Notations

- MPC/SMC: secure multi-party computation, or secure computation among two or more participants
- SFE: secure function evaluation
  - often used to mean the same as MPC, or
  - one party provides inputs to a function that is evaluated by an outsourced server
- 2PC: MPC between two parties
  - two-party MPC is an important special case, which received a lot of targeted attention, and
  - two-party protocols are often significantly different from the general n-party case

Common Notations

- $\mathsf{Enc}_k(m)$: encryption of a message m under key k
- $\mathsf{Dec}_k(m)$: decryption of a message m under key k
- $P_1, \ldots, P_n$: n participants, parties or players
- $\mathcal{A}$: an adversary

Secure Channels

- We assume existence of direct secure channels between each pairs of participating players
- Such channels could be achieved inexpensively through a variety of means, and are out of scope in this book

Negligible Function

- $v : \mathbb{N} \rightarrow \mathbb{R}$
- Any function that approaches zero asymptotically faster than any inverse polynomial
- For any polynomial p, $v(n) < \frac{1}{p(n)}$ for all but finitely many n

Computation and Statistical Security

- We will denote computational and statistical security parameters by κ and σ respectively
- κ governs the hardness of problems that can be broken by an adversary's offline computation
  - e.g., break an encryption scheme
- In practice, κ is typically set to a value like 128 or 256

Computation and Statistical Security

- Even considering security against computationally bounded adversaries, there may be attacks against an interactive protocol, not made easier by offline computation
- The interactive nature of a protocol may give the adversary only a single opportunity to violate security
  - e.g., by sending a message that has a special property, like predicting the random value that an honest party will chose in the next round

Computation and Statistical Security

- The statistical security parameter σ governs the hardness of these attacks
- In practice, σ is typically set to a smaller value like 40 or 80
- The correct way to interpret the presence of two security parameters is that security is violated only with probability

  $$2^{-\sigma} + v(\kappa)$$

  where v is a negligible function that depends on the resources of the adversary
- When we consider computationally unbounded adversaries, we omit κ and require $v = 0$

Random Sampling

- $\in_R$ denotes uniformly random sampling from a distribution
- For example, "choose $k \in_R \{0,1\}^\kappa$" means that k is a uniformly chosen κ-bit long string
- More generally, "$v \in_R D$" denotes sampling according to a probability distribution D
- Often the distribution is the output of a randomized algorithm
  - "$v \in_R A(x)$" denotes that v is the result of running randomized algorithm A on input x

Computational and Statistical Indistinguishability

- Let D_1 and D_2 be two probability distributions indexed by a security parameter
- D_1 and D_2 are indistinguishable if for all algorithms A there exists a negligible function v such that:

$$\Pr[A(D_1(n)) = 1] - \Pr[A(D_2(n)) = 1] \leq v(n)$$

- In other words, no algorithm behaves more than negligibly differently when given inputs sampled according to D_1 vs. D_2

Computational and Statistical Indistinguishability

- When we consider only non-uniform, polynomial-time algorithms A, the definition results in **computational indistinguishability**
- When we consider all algorithms without regard to their computational complexity, we get a definition of **statistical indistinguishability**
- In that case, the probability above is bounded by the statistical distance (also known as total variation distance) of the two distributions, which is defined as:

$$\Delta(D_1(n), D_2(n)) = \frac{1}{2}\sum_x |\Pr[A(D_1(n)) = 1] - \Pr[A(D_2(n)) = 1]|$$

Computational and Statistical Indistinguishability

- **Computational security** refers to security against adversaries implemented by non-uniform polynomial-time algorithms
- **Information-theoretic** security (also known as unconditional or statistical security) means security against arbitrary adversaries (even those with unbounded computational resources)

Secret Sharing

- Secret sharing is an essential primitive, that is at the core of many MPC approaches
- Informally, a (t, n)-secret sharing scheme splits the secret s into n shares, such that
  - any $t - 1$ of the shares reveal no information about s
  - any t shares allow complete reconstruction of s

Secret Sharing - Chor, 1993

Let D be the domain of secrets and D_1 be the domain of shares

- $\mathrm{Shr} : D \rightarrow D_1^n$ be a (possibly randomized) sharing algorithm
- $\mathrm{Rec} : D_1^k \rightarrow D$ be a reconstruction algorithm

A (t, n)-secret sharing scheme is a pair of algorithms (Shr, Rec) that satisfies these two properties:

Secret Sharing - Chor, 1993

1. **Correctness**: Let $(s_1, \ldots, s_n) = \mathrm{Shr}(s)$. Then

$$\Pr[\forall k \geq t, \mathrm{Rec}(s_{i_1}, \ldots, s_{i_k}) = s] = 1$$

2. **Perfect Privacy**: Any set of shares of size less than t does not reveal anything about the secret in the information theoretic sense. More formally, for any two secrets $a, b \in D$ and any possible vector of shares $v = v_1, \ldots, v_k$ where $k < t$

$$\Pr[v = \mathrm{Shr}(a)|_k] = \Pr[v = \mathrm{Shr}(b)|_k]$$

where $|_k$ denotes projection on a subspace of k elements

Secret Sharing

- Many discussions in this book use (n, n)-secret sharing schemes, where all n shares are necessary and sufficient to reconstruct the secret
- In some papers or books, (t, n)-secret sharing means
  - Any t shares cannot reconstruct the secret
  - Any $t + 1$ shares can completely reconstruct the secret
  - In this case, $t = n - 1$ at maximum

Random Oracle

- Random Oracle (RO) is a heuristic model for the security of hash functions, introduced by Bellare and Rogaway (1993)
  - The idea is to treat the hash function as a public, idealized random function
- In the random oracle model, all parties have access to the public function $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$, implemented as a stateful oracle

Random Oracle

- On input string $x \in \{0,1\}^*$, H looks up its history of calls
  - If $H(x)$ had never been called, H chooses a random $r_x \in \{0,1\}^\kappa$, remembers the pair (x, r_x) and returns r_x
  - If $H(x)$ had been called before, H returns r_x
- In this way, the oracle realizes a randomly-chosen function $\{0,1\}^* \to \{0,1\}^\kappa$

Random Oracle

- The RO model is a heuristic model, because it captures only those attacks that treat the hash function H as a black-box
- It deviates from reality in that it models a public function (e.g., a standardized hash function like SHA-256) as an inherently random object
- It is possible to construct schemes that are secure in the random oracle model, but insecure whenever H is instantiated by any concrete function (Canetti et al., 1998)

Random Oracle

- Despite these shortcomings, the random oracle model is often considered acceptable for practical applications
- Assuming a random oracle often leads to significantly more efficient constructions
- We will be careful to state when a technique relies on the random oracle model

Outline

Security of MPC

- Informally, the goal of MPC is for a group of participants to learn the correct output of some agreed-upon function applied to their private inputs without revealing anything else
- We now provide a more formal definition to clarify the security properties MPC aims to provide
- First, we present the real-ideal paradigm which forms the conceptual core of defining security
- Then we discuss two different but commonly used adversary models for MPC
- Finally, we discuss issues of composition - namely, whether security preserved in the natural way when a secure protocol invokes another subprotocol

How to Define Security

- A natural way to define security is to come up with a list of things that constitute a violation of security, e.g., the adversary should not be able to
  - learn a certain predicate of another party's input,
  - induce impossible outputs for the honest parties, or
  - make its inputs depend on honest parties' inputs
- This is tedious, cumbersome and error-prone
- It is not obvious when the list could be considered complete

How to Define Security

- The real-ideal paradigm avoids this pitfall completely by
  - introducing an "ideal world" that implicitly captures all security guarantees
  - defining security in relation to this ideal world
- The definition of probabilistic encryption by Goldwasser and Micali (1984) is considered to be the first instance of using this approach to define and prove security

Ideal World

- In the ideal world, the parties securely compute the function $\mathcal{F}$ by privately sending their inputs to a completely trusted party $\mathcal{T}$, referred to as the functionality
- Each party P_i has an input x_i, which is sent to $\mathcal{T}$ who simply computes $\mathcal{F}(x_1, \ldots, x_n)$ and returns the result to all parties
- Often we will make a distinction between $\mathcal{F}$ as a trusted party (functionality) and the circuit C that a party computes on the private inputs

Ideal World

- We can imagine an adversary attempting to attack the ideal-world interaction
  - An adversary can take control over any P_i, but not $\mathcal{T}$
- The simplicity of the ideal world makes it easy to understand the effect of such an attack
- Considering the previous attack list:
  - the adversary clearly learns no more than $\mathcal{F}(x_1, \ldots, x_n)$ since that is the only message it receives
  - the outputs given to the honest parties are all consistent
  - the adversary's choice of inputs is independent of the honest parties'

Ideal World

- The ideal world is easy to understand, but the presence of TTP makes it imaginary or impractical
- The ideal world serves as a benchmark against which to judge the security of an actual protocol

Real World

- In the real world, there is no trusted party, and all parties communicate with each other using a protocol
- The protocol π specifies for each party P_i a "next-message" function π_i
- π_i takes as input a security parameter, the party's private input x_i, a random tape, and the list of messages P_i has received
- Then, π_i outputs a next message to send with its destination or instructs the party to terminate with some specific output

Real World

- In the real world, an adversary can corrupt parties
  - corruption at the beginning of the protocol is equivalent to the original party being an adversary
- Depending on the threat model, corrupt parties may either follow the protocol as specified, or deviate arbitrarily in their behavior
- Intuitively, the real world protocol π is considered secure if any effect that an adversary can achieve in the real world can also be achieved by a corresponding adversary in the ideal world
- Put differently, the goal of a protocol is to provide security in the real world (given a set of assumptions) that is equivalent to that in the ideal world

Semi-Honest Security

- A **semi-honest** adversary is one who corrupts parties but follows the protocol as specified
- In other words, the corrupt parties run the protocol honestly but they may try to learn as much as possible from the messages they receive from other parties
- Note that this may involve several colluding corrupt parties pooling their views together in order to learn information
- Semi-honest adversaries are also considered **passive** in that they cannot take any actions other than attempting to learn private information by observing a **view** of a protocol execution
- Semi-honest adversaries are also commonly called **honest-but-curious**

View of a Protocol Execution

- The view of a party consists of its private input, its random tape, and the list of all messages received during the protocol
- The view of an adversary consists of the combined views of all corrupt parties
- Anything an adversary learns from running the protocol must be an efficiently computable function of its view

Simulation-based Security

- Following the real-ideal paradigm, security means that such an "attack" can also be carried out in the ideal world
- That is, for a protocol to be secure, it must be possible in the ideal world to generate something indistinguishable from the real world adversary's view
- Note that the adversary's view in the ideal world consists of nothing but inputs sent to $\mathcal{T}$ and outputs received from $\mathcal{T}$

Simulation-based Security

- Thus, an ideal-world adversary must be able to use this information to generate what looks like a real-world view
- This ideal-world adversary is referred as a simulator, since it generates a "simulated" real-world view while in the ideal-world
- Showing that such a simulator exists proves that there is nothing an adversary can accomplish in the real world that could not also be done in the ideal world

Simulation-based Security

More formally, let π be a protocol and $\mathcal{F}$ be a functionality. Let C be the set of parties that are corrupted, and let Sim denote a simulator algorithm. We define the following distributions of random variables:

- $\mathrm{Real}_\pi(\kappa, C; x_1, \ldots, x_n)$: run the protocol with security parameter κ, where each party P_i runs the protocol honestly using private input x_i. Let V_i denote the final view of party P_i, and let y_i denote the final output of party P_i

  Output $(\{V_i | i \in C\}, (y_1, \ldots, y_n))$

- $\mathrm{Ideal}_{\mathcal{F},\mathrm{Sim}}(\kappa, C; x_1, \ldots, x_n)$: $\mathrm{Compute}(y_1, \ldots, y_n) \leftarrow \mathcal{F}(x_1, \ldots, x_n)$

  Output $(\mathrm{Sim}(C, \{(x_i, y_i) | i \in C\}), (y_1, \ldots, y_n))$

Simulation-based Security

A protocol is secure against semi-honest adversaries if the corrupted parties in the real world have views that are indistinguishable from their views in the ideal world:

Definition (Semi-Honest Security)

A protocol π securely realizes $\mathcal{F}$ in the presence of semi-honest adversaries if there exists a simulator Sim such that, for every subset of corrupt parties C and all inputs $x_1, \ldots, x_n$

- the distributions $\mathrm{Real}_\pi(\kappa, C; x_1, \ldots, x_n)$ and $\mathrm{Ideal}_{\mathcal{F},\mathrm{Sim}}(\kappa, C; x_1, \ldots, x_n)$ are indistinguishable in κ

Additional Observations

- In defining Real and Ideal we have included the outputs of all parties, even the honest ones
  - to incorporate a correctness condition into the definition
- In the case that no parties are corrupt ($C = \emptyset$)
  - The output of Real and Ideal simply consists of all parties' outputs in the two interactions
  - Thus, the definition implies that protocol gives outputs distributed just as their outputs from the ideal functionality

Why Semi-Honest Security

- The semi-honest adversary model may seem exceedingly weak
  - simply reading and analyzing received messages barely even seems like an attack at all
- Achieving semi-honest security is far from trivial
- Semi-honest protocols often serve as a basis for protocols in more robust settings with powerful attackers
- Many realistic scenarios do correspond to semi-honest attack
  - E.g., computing with players who are trusted to act honestly, but cannot fully guarantee that their storage might not be compromised in the future

Malicious Security

- A **malicious** (also known as **active**) adversary may instead cause corrupted parties to deviate arbitrarily from the prescribed protocol in an attempt to violate security
- A malicious adversary has all the powers of a semi-honest one, but may also take any actions it wants during protocol execution
- This subsumes an adversary that can control, manipulate, and arbitrarily inject messages on the network (even through we assume direct secure channels between each pair of parties)
- Security in this setting is also defined in comparison to the ideal world, but there are two important additions to consider: **effect on honest outputs** and **extraction**

Malicious Security - Effect on Honest Outputs

- When the corrupt parties deviate from the protocol, there is now the possibility that honest parties' outputs will be affected
  - E.g., imagine an adversary that causes two honest parties to output different things while in the ideal world all parties get identical outputs
- This condition is somewhat trivialized in the previous definition that does compare real-world outputs to ideal-world outputs
- Furthermore, we can/should make no guarantees on the final outputs of corrupt parties, only of the honest parties, since a malicious party can output whatever it likes

Malicious Security - Extraction

- Honest parties follow the protocol according to a well-defined input, which can be given to $\mathcal{T}$ in the ideal world as well
- In contrast, the input of a malicious party is not well-defined in the real world, which leads to the question of what input should be given to $\mathcal{T}$ in the ideal world
- Intuitively, in a secure protocol, whatever an adversary can do in the real world should also be achievable in the ideal world by some suitable choice of inputs for the corrupt parties

Malicious Security - Extraction

- Hence, we leave it to the simulator to choose inputs for the corrupt parties
- This aspect of simulation is called extraction, since the simulator extracts an effective ideal-world input from the real-world adversary that "explains" the input's real-world effect
- In most constructions, it is sufficient to consider black-box simulation, where the simulator is given access only to the oracle implementing the real-world adversary, and not its code

Malicious Security

- When $\mathcal{A}$ denotes the adversary program, we write corrupt($\mathcal{A}$) to denote the set of parties that are corrupted
- corrupt(Sim) for the set of parties that are corrupted by the ideal adversary, Sim
- Similar to semi-honest definition, we define distributions for the real world and ideal world, and define a secure protocol as one that makes those distributions indistinguishable

Malicious Security

- $\mathrm{Real}_{\pi,\mathcal{A}}(\kappa; \{x_i | i \notin \mathrm{corrupt}(\mathcal{A})\})$: run the protocol with security parameter κ, where each honest party P_i (for $i \notin \mathrm{corrupt}(\mathcal{A})$) runs the protocol honestly using private input x_i, and the messages of corrupt parties are chosen according to $\mathcal{A}$. Let y_i denote the output of each honest party P_i and let V_i denote the final view of party P_i

  Output $(\{V_i | i \in \mathrm{corrupt}(\mathcal{A})\}, \{y_i | i \notin \mathrm{corrupt}(\mathcal{A})\})$

- $\mathrm{Ideal}_{\mathcal{F},\mathrm{Sim}}(\kappa; \{x_i | i \notin \mathrm{corrupt}(\mathcal{A})\})$: Rum Sim until it outputs a set of inputs $\{x_i | i \in \mathrm{corrupt}(\mathcal{A})\}$. $\mathrm{Compute}(y_1, \ldots, y_n) \leftarrow \mathcal{F}(x_1, \ldots, x_n)$. Then, give $\{y_i | i \in \mathrm{corrupt}(\mathcal{A})\}$ to Sim. Let V^* denote the final output of Sim (a set of simulated views)

  Output $(V^*, \{y_i | i \notin \mathrm{corrupt}(\mathrm{Sim})\})$

Malicious Security

Definition (Malicious Security)

A protocol π securely realizes $\mathcal{F}$ in the presence of malicious adversaries if for every real-world adversary $\mathcal{A}$, there exists a simulator Sim with $\mathrm{corrupt}(\mathcal{A}) = \mathrm{corrupt}(\mathrm{Sim})$, such that, for all inputs for honest parties $\{x_i | i \notin \mathrm{corrupt}(\mathcal{A})\}$

- the distributions $\mathrm{Real}_{\pi,\mathcal{A}}(\kappa; \{x_i | i \notin \mathrm{corrupt}(\mathcal{A})\})$ and $\mathrm{Ideal}_{\mathcal{F},\mathrm{Sim}}(\kappa; \{x_i | i \notin \mathrm{corrupt}(\mathcal{A})\})$ are indistinguishable in κ

Malicious Security

- The definition quantifies only over the inputs of honest parties
- The interaction `Real` does not consider the corrupt parties to have any inputs, and the inputs of the corrupt parties in `Sim` is only determined indirectly (by the simulator's choice of what to send to $\mathcal{F}$ on the corrupt parties' behalf)
- It is possible to also inputs for corrupt parties in the real world, such inputs would merely be "suggestions" since corrupt parties could choose to run the protocol on any other input

Reactive functionalities

- In the ideal world, the interaction with the functionality consists of just a single round: inputs followed by outputs
- It is possible to generalize the behavior of $\mathcal{F}$ so that it interacts with the parties over many rounds of interaction, keeping its own private internal state between rounds
- Such functionalities are called `reactive`

Reactive functionalities - Examples

1. The dealer in a poker game
   - The functionality must keep track of the state of all cards
   - Taking input commands and giving outputs to all parties in many rounds
2. Commitment
   - This functionality accepts a bit b (or more generally, a string) from P_1 and gives output "committed" to P_2, while internally remembering b
   - At some later time, if P_1 sends the command "reveal" (or "open") to the functionality, it gives b to P_2

Security with Abort

- In any message-based two-party protocol, one party will learn the final output before the other
- If that party is corrupt and malicious, they may simply refuse to send the last message to the honest party and thereby prevent the honest party from learning the output
- However, this behavior is incompatible with our previous description of the ideal world
  - If corrupt parties receive output from the functionality then all parties do
  - This property is called output fairness and not all functions can be computed with this property (Cleve, 1986; Gordon et al., 2008; Asharov et al., 2015a)

Security with Abort

- Typical results in the malicious setting provide a weaker property known as `security with abort`, which requires slightly modifying the ideal functionality as follows
- First, the functionality is allowed to know the identities of the corrupt parties
- The functionality's behavior is modified to be slightly reactive

Security with Abort - Reactive Functionality

1. After all parties have provided input, the functionality computes outputs and delivers the outputs to the corrupt parties only
2. Then the functionality awaits either a `deliver` or `abort` command from the corrupted parties
   - Upon receiving `deliver`, the functionality delivers the outputs to all the honest parties
   - Upon receiving `abort`, the functionality delivers an abort output ($\perp$) to all the honest parties

Security with Abort

- In this modified ideal world, an adversary is allowed to learn the output before the honest parties and to prevent the honest parties from receiving any output
- It is important to note, however, that whether an honest party aborts can depend only on the corrupt party's outputs
- In particular, it would violate security if the honest party's abort probability to be depended on its own input

Security with Abort

- Usually the possibility of blocking outputs to honest parties is not written explicitly in the description of the functionality
- It is generally understood that when discussing security against malicious adversaries, the adversary has control over output delivery to honest parties and output fairness is not expected

Adaptive Corruption

- **Static corruption**: the identities of the corrupted parties are fixed throughout the entire interaction
  - This provides security against static corruption
- **Adaptive corruption**: an adversary may choose which parties to corrupt during the protocol execution, possibly based on what it learns during the interaction
  - This behavior is known as adaptive corruption
- This book considers only static corruption, following the vast majority of work in the field

Hybrid Worlds and Composition

- In the interest of modularity, it is often helpful to design protocols that make use of other ideal functionalities
- E.g., design a protocol π that securely realizes some functionality $\mathcal{F}$, where the parties of π also interact with another functionality $\mathcal{G}$ in addition to sending messages to each other
- Hence, the real world for this protocol includes $\mathcal{G}$, while the ideal world (as usual) includes only $\mathcal{F}$.
- We call this modified real world the $\mathcal{G}$-hybrid world

Hybrid Worlds and Composition

- A natural requirement for a security model is **composition**:
  - if π is a $\mathcal{G}$-hybrid protocol that securely realizes $\mathcal{F}$ (i.e., parties in π send messages and also interact with an ideal $\mathcal{G}$), and ρ is a protocol that securely realizes $\mathcal{G}$
  - then composing π and ρ in the natural way (replacing every invocation of $\mathcal{G}$ with a suitable invocation of ρ) also results in a secure protocol for $\mathcal{F}$
- It may be surprising that some very natural ways of specifying the details do not guarantee composability of secure protocols

Hybrid Worlds and Composition

- The standard way to achieve guaranteed composition is to use **universal composability** (UC) framework from Canetti (2001)
- The UC framework augments the security model that we have sketched here with an additional entity called the environment, which is included in both the ideal and real worlds
- The goal of the environment is to capture the "context" in which the protocol executes (e.g., the protocol under consideration is invoked as a small step in some larger calling protocol)
- The environment chooses inputs for the honest party and receives their outputs
- It also may interact arbitrarily with the adversary

Outline

Oblivious Transfer

- Oblivious Transfer (OT) is an essential building block for secure computation protocols
- It is theoretically equivalent to MPC as shown by Kilian (1988):
  - Given OT, one can build MPC without any additional assumptions
  - Similarly, one can directly obtain OT from MPC

Oblivious Transfer

- The standard definition of 1-out-of-2 OT involves two parties, a Sender S holding two secrets x_0, x_1, and a receiver R holding a choice bit $b \in \{0, 1\}$
- OT is a protocol allowing R to obtain x_b while learning nothing about the "other" secret x_{1-b}
- At the same time, S does not learn anything at all

OT Definition

Definition (Oblivious Transfer)

A 1-out-of-2 OT is a cryptographic protocol securely implementing the functionality $\mathcal{F}^{\mathrm{OT}}$ defined below:

Parameters:

- Two parties: Sender S and Receiver R. S has input secrets $x_0, x_1 \in \{0,1\}^n$, and R has a selection bit $b \in \{0,1\}$

Functionality:

- S sends x_0 and x_1 to $\mathcal{F}^{\mathrm{OT}}$, and R sends b to $\mathcal{F}^{\mathrm{OT}}$
- R receives x_b, and S receives $\perp$

OT Variants

- Many variants of OT may be considered
- A natural variant is 1-out-of-k OT:
  - S holds k secrets, and
  - R has a choice selector from $\{0, \ldots, k-1\}$
- Another invariant is t-out-of-k OT, where $2 \leq t < k$

Commitment

- Commitment is fundamental in many cryptographic protocols
- A commitment scheme allows a sender to commit to a secret value, and reveal it at some later time to a receiver
  - **Hiding property**: the receiver should learn nothing about the committed value before it is revealed by the sender
  - **Binding property**: the sender should not be able to change its choice of value after committing

Commitment

- Commitment is rather simple and inexpensive in the random oracle model
- To commit to x, simply choose a random value $r \in_R \{0,1\}^\kappa$ and publish the value $y = H(x||r)$
- To later reveal, simply announce x and r

Commitment Definition

Definition (Commitment)

Commitment is a cryptographic protocol securely implementing the functionality $\mathcal{F}^{\mathrm{Comm}}$ defined below:

Parameters:

- Two parties: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$. Length of committed string n

Functionality:

- $\mathcal{S}$ sends a string $s \in \{0,1\}^n$ $\mathcal{F}^{\mathrm{Comm}}$, and $\mathcal{F}^{\mathrm{Comm}}$ sends **committed** to $\mathcal{R}$
- At some later time, $\mathcal{S}$ sends **open** to $\mathcal{F}^{\mathrm{Comm}}$, and $\mathcal{F}^{\mathrm{Comm}}$ sends s to $\mathcal{R}$

Zero-Knowledge Proof

- A zero-knowledge (ZK) proof allows a prover to convince a verifier that it knows x such that $C(x) = 1$, without revealing any further information about x, and C is a public predicate

ZK-Proof Example

- Suppose G is a graph known to both Alice and Bob, and only Alice knows a 3-coloring $\mathcal{X}$ for G
- Then Alice can use a ZK proof to convince Bob that G is 3-colorable without disclosing $\mathcal{X}$ to Bob.
- She constructs a circuit C_G that interprets its input as an encoding of a 3-coloring and checks whether it is a legal 3-coloring of G
- She uses $(C_G, \mathcal{X})$ as input to the ZK proof

ZK-Proof Example

- From Bob's point of view, he receives output (**proven**, C_G) if and only if Alice was able to provide a valid 3-coloring of G
- At the same time, Alice knows that Bob learned nothing about her 3-coloring $\mathcal{X}$ other than the fact that some legal $\mathcal{X}$ exists

ZK Definition

Definition (ZK-Proof)

A zero-knowledge proof is a cryptographic protocol securely implementing the functionality $\mathcal{F}^{\mathrm{zk}}$ defined below:

Parameters:

- Two parties: Prover $\mathcal{P}$ and Verifier $\mathcal{V}$

Functionality:

- $\mathcal{P}$ sends a string (C, x) to $\mathcal{F}^{\mathrm{zk}}$, where $C : \{0,1\}^n \to \{0,1\}$ is a Boolean circuit with 1 output bit, and $x \in \{0,1\}^n$
- If $C(x) = 1$, then $\mathcal{F}^{\mathrm{zk}}$ sends (**proven**, C) to $\mathcal{V}$; otherwise, it sends $\perp$ to $\mathcal{V}$

Acknowledgment

The contents of these slides are based on the following book:

- A Pragmatic Introduction to Secure Multi-Party Computation
  `https://securecomputation.org/`
- Chapter 2: Defining Multi-Party Computation